Bridging LTL_f Inference to GNN Inference for Learning LTL_f Formulae

Weilin Luo[1], Pingjia Liang[1], Jianfeng Du[3,4*], Hai Wan[1,2*], Bo Peng[1], Delong Zhang[1]

[1] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
[2] Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-sen University), Ministry of Education, China
[3] Guangzhou Key Laboratory of Multilingual Intelligent Processing,
Guangdong University of Foreign Studies, Guangzhou, China
[4] Pazhou Lab, Guangzhou, China
{luowlin3,liangpj3,pengb8,zhangdlong3}@mail2.sysu.edu.cn, wanhai@mail.sysu.edu.cn, jfdu@gdufs.edu.cn

Abstract

Learning *linear temporal logic on finite traces* (LTL_f) formulae aims to learn a *target formula* that characterizes the high-level behavior of a system from observation traces in planning. Existing approaches to learning LTL_f formulae, however, can hardly learn accurate LTL_f formulae from noisy data. It is challenging to design an efficient search mechanism in the large search space in form of arbitrary LTL_f formulae while alleviating the wrong search bias resulting from noisy data. In this paper, we tackle this problem by bridging LTL_f inference to GNN inference. Our key theoretical contribution is showing that GNN inference can simulate LTL_f inference to distinguish traces. Based on our theoretical result, we design a GNN-based approach, `GLTLf`, which combines GNN inference and parameter interpretation to seek the target formula in the large search space. Thanks to the non-deterministic learning process of GNNs, `GLTLf` is able to cope with noise. We evaluate `GLTLf` on various datasets with noise. Our experimental results confirm the effectiveness of GNN inference in learning LTL_f formulae and show that `GLTLf` is superior to the state-of-the-art approaches.

Introduction

We examine the problem of learning *target formulae* that characterize the high-level behavior of a system from observation traces in planning. We focus on the *linear temporal logic on finite traces* (LTL_f) formula because it facilitates human interpretation and manipulation (Camacho and McIlraith 2019). Learning LTL_f formulae has wide applications, *e.g.*, verification of system properties (Kasenberg and Scheutz 2017), behavior classification (Camacho and McIlraith 2019), and explainable models (Kim et al. 2019), *etc.*

It is valuable to learn *arbitrary* LTL_f formulae from noisy data. In theory, arbitrary LTL_f formulae are of more perfect characterization and expressive ability than the formula restricted by LTL_f *templates*. Besides, in practice, the traces, *e.g.*, from sensors or unintended user behavior, usually contain noise (Kim et al. 2019). It is challenging to learn arbitrary LTL_f formulae from noisy data. Firstly, the search space of the target formula is huge because the target formula with arbitrary form only has semantic constraints rather than syntactic constraints. Secondly, noisy data tend to bring wrong search bias deviated from the target formula.

There have been some approaches attempting to address the above challenge. Some approaches, *e.g.*, (Neider and Gavran 2018; Camacho and McIlraith 2019), assume a noise-free environment. Other approaches, *e.g.*, (Lemieux, Park, and Beschastnikh 2015; Shah et al. 2018; Kim et al. 2019), restrict the hypothesis space by LTL_f templates. Recently, Gaglione et al. (2021) proposed a MaxSAT-based approach to learn arbitrary LTL_f formulae while dealing with noisy data. Their approaches, however, are subject to the high complexity of MaxSAT.

In this paper, we propose a novel way to tackle the above challenge, by bridging LTL_f inference to *Graph Neural Network* (GNN) inference. Intuitively, an LTL_f formula distinguishes positive traces (*i.e.*, the trace satisfies the formula) from negative traces (*i.e.*, the trace does not satisfy the formula), based on the satisfaction relations between traces and formulae. We discover that GNNs can capture the satisfaction relations. In more details, we show in Theorem 1 that GNN inference can simulate LTL_f inference to distinguish traces. It provides an intuition for exploiting learning a GNN classifier to search for approximate LTL_f formulae, *i.e.*, modeling the search problem in discrete space to the parameter learning problem in continuous space.

Based on this theoretical result, we design a GNN-based approach, named as `GLTLf`, which combines GNN inference and parameter interpretation to seek the target formula in the large search space. Specifically, `GLTLf` first learns a GNN classifier to distinguish traces. It then extracts an LTL_f formula by interpreting the parameters of the learned GNN classifier. Thanks to the non-deterministic learning process of GNN, `GLTLf` is good at handling noisy data.

Following the assessment methods of the state-of-the-art approaches (Camacho and McIlraith 2019; Kim et al. 2019), we evaluate `GLTLf` on traces with some noise across benchmarks. Experimental results confirm the highlights of `GLTLf`. Compared with other approaches, `GLTLf` is demonstrated stronger robustness for noisy data and better scalability in data size.

*Corresponding authors.

Preliminaries

Notation. A *graph* G is a pair (V, E), where V is a finite set of *vertices* and $E \subseteq \{\langle u, v\rangle | u, v \in V \wedge u \neq v\}$ is a set of *edges*. The edge is directed: if an edge $e = \langle u, v\rangle$, then e is from u to v. We denote the set of vertices (*resp.* edges) of G by $\mathsf{V}(G)$ (*resp.* $\mathsf{E}(G)$). Moreover, $\mathcal{N}(v)$ denotes the *neighborhood* of $v \in \mathsf{V}(G)$, formally, $\mathcal{N}(v) = \{u \in \mathsf{V}(G) | \langle u, v\rangle \in \mathsf{E}(G)\}$. A *path graph* is a graph whose vertices can be listed in the order $v_0, v_1, \ldots, v_n$ such that the edges are $\langle v_i, v_{i+1}\rangle$ where $i = 0, 1, \ldots, n-1$. Let $\{\{\ldots\}\}$ denotes a multiset. Let $(\mathbf{v})_i$ denote the i^{th} element of a vector $\mathbf{v}$. Let $(\mathbf{A})_{ij}$ denote the element of a matrix $\mathbf{A}$ at the i^{th} row and the j^{th} column.

Graph Neural Networks. *Graph neural networks (GNNs)* (Hamilton, Ying, and Leskovec 2017) are a powerful neural architecture to learn vector representations of vertices and graphs. A GNN model consists of a stack of neural network layers, where each layer aggregates local neighborhood information, *i.e.*, features of neighbors, and then combines with the feature of itself to generate the feature of the next layer. Formally, a *aggregate-combine GNN (AC-GNN)* model is defined as follows. Let G be a graph. Each vertex $v \in \mathsf{V}(G)$ has an associated feature vector $\mathbf{x}_v$. $\mathbf{x}_v^{(0)}$ is an initial representation. In each layer $t > 0$, the updating representation $\mathbf{x}_v^{(t)}$ is defined as follows:

$$\mathbf{x}_v^{(t)} = \mathsf{COM}^{(t)}(\mathbf{x}_v^{(t-1)}, \mathsf{AGG}^{(t)}(\{\{\mathbf{x}_u^{(t-1)} | u \in \mathcal{N}(v)\}\})),$$

where $\mathsf{AGG}^{(t)}$ and $\mathsf{COM}^{(t)}$ are *aggregation* and *combination* functions, respectively. We call an AC-GNN *simple* if

$$\mathsf{AGG}(X) = \sum_{\mathbf{x} \in X} \mathbf{x},$$

$$\mathsf{COM}^{(t)}(\mathbf{x}_1, \mathbf{x}_2) = \sigma(\mathbf{C}^{(t)}\mathbf{x}_1 + \mathbf{A}^{(t)}\mathbf{x}_2 + \mathbf{b}^{(t)}),$$

where $\mathbf{C}^{(t)}$, $\mathbf{A}^{(t)}$, and $\mathbf{b}^{(t)}$ are parameters and σ denotes a component-wise non-linear function, *e.g.*, sigmoid or ReLU. We call an AC-GNN *homogeneous* if it shares the same parameters across layers. Finally, each vertex $v \in \mathsf{V}(G)$ is classified according to a Boolean classification function applied to $\mathbf{x}_v^{(T)}$, where $T > 0$ denotes the last layer.

Linear temporal logic. *Linear temporal logic (LTL)* is a modal logic typically used to express temporally extended constraints over state trajectories (Pnueli 1977). Derived from LTL, LTL interpreted over finite traces is referred to as *finite LTL (LTL$_f$)* (Baier and McIlraith 2006; Giacomo and Vardi 2013). The syntax of LTL$_f$ for a finite set of atomic propositions $\mathbb{P}$ includes $\top$ (true) and $\bot$ (false), standard logical operators ($\wedge$ and $\neg$), and temporal logical operators: *next* (X) and *until* (U), described as follows:

$$\phi := \bot \mid \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathsf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2,$$

where $p \in \mathbb{P} \cup \{\top, \bot\}$ and φ, φ_1, and φ_2 are LTL$_f$ formulae.

Operator *or* ($\vee$), *weak next* (N), *release* (R), *eventually* (F), and *always* (G) are commonly used, and can be defined as $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathsf{N}\varphi := \neg\mathsf{X}\top \vee \mathsf{X}\varphi$, $\varphi_1 \mathsf{R} \varphi_2 := \neg(\neg\varphi_1 \mathsf{U} \neg\varphi_2)$, $\mathsf{F}\varphi := \top \mathsf{U} \varphi$, and $\mathsf{G}\varphi := \neg(\top \mathsf{U} \neg\varphi)$ respectively. For brevity, we only consider the fundamental operators in this paper. Our approach allows to receive all temporal modal operators of LTL$_f$. The *size* of an LTL$_f$ formula ϕ is the number of logical operators and atomic propositions in ϕ, denoted by $|\phi|$.

Definition 1. *Let ϕ be an LTL$_f$ formula. The set of* sub-formulae $\mathsf{sub}(\phi)$ *of ϕ is defined recursively as follows:*

- *if $\phi = p$, then* $\mathsf{sub}(\phi) = \{p\}$*;*
- *if $\phi = \mathsf{o}_1\ \phi'$, then* $\mathsf{sub}(\phi) = \{\phi\} \cup \mathsf{sub}(\phi')$*;*
- *if $\phi = \phi'\ \mathsf{o}_2\ \phi''$, then* $\mathsf{sub}(\phi) = \{\phi\} \cup \mathsf{sub}(\phi') \cup \mathsf{sub}(\phi'')$*,*

where $p \in \mathbb{P} \cup \{\top, \bot\}$, $\mathsf{o}_1 \in \{\neg, \mathsf{X}\}$, $\mathsf{o}_2 \in \{\wedge, \mathsf{U}\}$, and ϕ', ϕ'' are LTL$_f$ formulae.

LTL$_f$ formulae are interpreted over finite *traces* of propositional states. An finite trace is represented in the form $\pi = s_0, s_1, \ldots, s_n$, where $s_t \in 2^{\mathbb{P}}$ is a state at time t. For every state s_i of π and every $p \in \mathbb{P}$, p holds if $p \in s_i$ or $\neg p$ holds otherwise. The traces mentioned in this paper are finite. The size of π is the number of states of π, denoted by $|\pi|$. π_i denotes a *sub-trace* of π beginning from the state s_i. Let π is a trace and $|\pi| = n$. The *satisfaction relation* $\models$ is defined as follows:

$\pi_i \models p$	iff	$p \in s_i$
$\pi_i \models \neg\phi$	iff	$\pi_i \not\models \phi$
$\pi_i \models \phi_1 \wedge \phi_2$	iff	$\pi_i \models \phi_1$ and $\pi_i \models \phi_2$
$\pi_i \models \mathsf{X}\phi$	iff	$i < n$ and $\pi_{i+1} \models \phi$
$\pi_i \models \phi_1 \mathsf{U} \phi_2$	iff	$\exists i \leq k \leq n, \pi_k \models \phi_2$ and $\forall i \leq j < k, \pi_j \models \phi_1$.

where ϕ, ϕ_1, ϕ_2 are LTL$_f$ formulae, and $p \in \mathbb{P} \cup \{\top, \bot\}$.

Learning LTL$_f$ formulae. We focus on the problem of learning a *target LTL$_f$ formula* to characterize the behaviors observed in a set of *positive traces* (Π^+), while to exclude behaviors observed in a set of *negative traces* (Π^-). The target formula is an *LTL$_f$ classifier over traces* aiming to separate the provided positive and negative traces. We denote $\Pi = \Pi^+ \cup \Pi^-$. We define $\mathsf{lab}\colon \Pi \to \{0, 1\}$: for any $e \in \Pi$, $\mathsf{lab}(e) = 1$ if and only if $e \in \Pi^+$. The *accuracy* of an LTL$_f$ formula ϕ for Π is defined as follows:

$$\mathsf{acc}(\phi, \Pi) = \frac{|\{\pi \in \Pi^+ | \pi \models \phi\}| + |\{\pi \in \Pi^- | \pi \not\models \phi\}|}{|\Pi|}$$

Related Work

Learning temporal logic formulae. Temporal logic formula, such as LTL and LTL$_f$ formula, is interpretable and expressive. So learning temporal logic formula has been verified very useful in many applications. Some works focus on learning formulae from positive and negative traces. Kim et al. (2019) proposed a method based on bayesian inference to handle noisy data, but limited in the hypothesis space of formulae due to using templates. Neider and Gavran (2018) proposed two novel learning algorithms for LTL formulae, one based on SAT solving, another based on learning decision trees. Further, Camacho and McIlraith (2019) designed a SAT-based method to learn formulae from a symbolic state representation of arbitrary LTL$_f$ formulae. Both (Neider and Gavran 2018) and (Camacho and McIlraith 2019) are limited in a noise-free environment. Gaglione et al. (2021) proposed a MaxSAT-based approach to extract concise LTL$_f$ formulae in a noise environment, however, the scalability of which is limited in calling the MaxSAT solver.

Other approaches learn temporal logic formulae from other forms of data. Kasenberg and Scheutz (2017) inferred formulae from Markove Decision Processes. Some approaches only focus on positive traces (la Rosa and McIlraith 2011; Lemieux, Park, and Beschastnikh 2015; Le and Lo 2015; Shah et al. 2018; Cao et al. 2018). In business process modelling, some works (Maggi, Bose, and van der Aalst 2012; Ciccio, Mecella, and Mendling 2013; Maggi et al. 2018; Leno et al. 2020) learned process models using temporal logic from event logs, a kind of positive traces. Recently, Maggi, Montali, and Peñaloza (2020) modelled uncertainty, *i.e.*, the noise, using probabilistic temporal logic.

Learning other logic representation. There exists many works on learning regular languages, such as deterministic finite-state automata (Angluin 1987; Giantamidis and Tripakis 2016), non-deterministic automata (Bollig et al. 2009), alternating automata (Angluin, Eisenstat, and Fisman 2015), finite-state machines (Smetsers, Fiterau-Brostean, and Vaandrager 2018). However, these approaches use explicit state representations and also suffer from scalability.

Other approaches include signal temporal logic (Asarin et al. 2011; Jin et al. 2015; Bombara et al. 2016; Xu et al. 2016; Xu and Julius 2016; Kong, Jones, and Belta 2016), past time linear temporal logic (Arif et al. 2020), interval temporal logic (Brunello, Sciavicco, and Stan 2020), property specification language (Roy, Fisman, and Neider 2020), and graph temporal logic (Xu et al. 2019b), *etc.*

Expressiveness of GNNs. The ability of GNNs has been recently characterized in the Weisfeiler-Lehman test (Xu et al. 2019a; Morris et al. 2019) or the logic FOC_2, a fragment of first order logic (Barceló et al. 2020). These studies inspire us to explore the relationship between GNNs and LTL_f.

Relationship between GNNs and LTL_f

In the following, we explore the relationship between GNNs and LTL_f. Based on the satisfaction relation of LTL_f, we have Property 1.

Property 1. *Let ϕ be an LTL_f formula and π a trace. For any sub-trace π_i of π, it fulfills following property:*

- *if $\phi = p$, then $\pi_i \models \phi$ if and only if $\pi_i \models p$;*
- *if $\phi = \neg\phi_1$, then $\pi_i \models \phi$ if and only if $\pi_i \not\models \phi_1$;*
- *if $\phi = \mathsf{X}\phi_1$, then $\pi_i \models \phi$ if and only if $\pi_{i+1} \models \phi_1$;*
- *if $\phi = \phi_1 \wedge \phi_2$, then $\pi_i \models \phi$ if and only if $\pi_i \models \phi_1$ and $\pi_i \models \phi_2$;*
- *if $\phi = \phi_1 \mathsf{U} \phi_2$, then $\pi_i \models \phi$ if and only if it fulfills two conditions: (1) $\pi_i \models \phi_2$ or $\pi_i \models \phi_1$; (2) $\pi_i \models \phi_2$ or $\pi_{i+1} \models \phi$;*

where ϕ, ϕ_1, ϕ_2 are LTL_f formulae, and $p \in \mathbb{P} \cup \{\top, \bot\}$.

Property 1 shows that the satisfaction relation between a sub-trace π_i and an LTL_f formula ϕ depends on the satisfiability of the sub-formulae of the current state (s_i) or the satisfiability of the sub-formulae of the next state (s_{i+1}). Therefore, we explore the relation between GNN and LTL_f by constructing the relation of satisfiability of formulae between current state and the next state.

Definition 2. *Let ϕ be an LTL_f formula. The* semantics unfolding *of ϕ, denoted by* $\mathsf{unfold}(\phi)$*, and its* semantics elements *of ϕ, denoted by* $\mathsf{element}(\phi)$*, are defined as follows:*

- *if $\phi = p$, then* $\mathsf{unfold}(\phi) = p$ *and* $\mathsf{element}(\phi) = \{p\}$*;*
- *if $\phi = \neg\phi_i$, then* $\mathsf{unfold}(\phi) = \neg\phi_i$ *and* $\mathsf{element}(\phi) = \{\phi_i\}$*;*
- *if $\phi = \phi_i \wedge \phi_j$, then* $\mathsf{unfold}(\phi) = \phi_i \wedge \phi_j$ *and* $\mathsf{element}(\phi) = \{\phi_i, \phi_j\}$*;*
- *if $\phi = \mathsf{X}\phi_i$, then* $\mathsf{unfold}(\phi) = \mathsf{X}\phi_i$ *and* $\mathsf{element}(\phi) = \{\mathsf{X}\phi_i\}$*;*
- *if $\phi = \phi_i \mathsf{U} \phi_j$, then* $\mathsf{unfold}(\phi) = \phi_j \vee (\phi_i \wedge \mathsf{X}\phi)$ *and* $\mathsf{element}(\phi) = \{\phi_i, \phi_j, \mathsf{X}\phi\}$*,*

where $p \in \mathbb{P} \cup \{\top, \bot\}$ and ϕ_i, ϕ_j are LTL_f formulae.

Property 2. *Let ϕ be an LTL_f formula. For any $\phi_i \in \mathsf{sub}(\phi)$,* $\mathsf{element}(\phi_i) \subseteq \{\phi_j, \mathsf{X}\phi_j | \phi_j \in \mathsf{sub}(\phi)\}$ *holds.*

Property 2 ensures that we can construct relation between ϕ and $\mathsf{element}(\phi)$ shown in Definition 3.

Definition 3. *Let ϕ be an LTL_f formula. Its* LTL_f graph *G_ϕ is a four-tuple $(V_\phi, E_\phi, \mathsf{W}_\phi, \mathsf{b}_\phi)$ defined as follows, where V_ϕ is a set of vertex, $E_\phi \subseteq V_\phi \times V_\phi$, $\mathsf{W}_\phi \colon E_\phi \to \mathbb{N}$, and $\mathsf{b}_\phi \colon V_\phi \to \mathbb{N}$. V_ϕ and E_ϕ are initialized as $\{v_\phi\}$ and $\emptyset$, respectively. For each sub-formula $\phi_i \in \mathsf{sub}(\phi)$, V_ϕ, E_ϕ, W_ϕ, and b_ϕ iteratively are constructed as follows:*

- *if* $\mathsf{unfold}(\phi_i) = p$*, then* $V_\phi = V_\phi \cup \{v_p\}$, $E_\phi = E_\phi \cup \{\langle v_p, v_p\rangle\}$, $\mathsf{W}_\phi(\langle v_p, v_p\rangle) = 1$, *and* $\mathsf{b}_\phi(v_p) = 0$*;*
- *if* $\mathsf{unfold}(\phi_i) = \neg\phi_j$*, then* $V_\phi = V_\phi \cup \{v_{\phi_j}\}$, $E_\phi = E_\phi \cup \{\langle v_{\phi_j}, v_{\phi_i}\rangle\}$, $\mathsf{W}_\phi(\langle v_{\phi_j}, v_{\phi_i}\rangle) = -1$, *and* $\mathsf{b}_\phi(v_{\phi_i}) = 1$*;*
- *if* $\mathsf{unfold}(\phi_i) = \phi_j \wedge \phi_k$*, then* $V_\phi = V_\phi \cup \{v_{\phi_j}, v_{\phi_k}\}$, $E_\phi = E_\phi \cup \{\langle v_{\phi_j}, v_{\phi_i}\rangle, \langle v_{\phi_k}, v_{\phi_i}\rangle\}$, $\mathsf{W}_\phi(\langle v_{\phi_j}, v_{\phi_i}\rangle) = 1$, $\mathsf{W}_\phi(\langle v_{\phi_k}, v_{\phi_i}\rangle) = 1$, *and* $\mathsf{b}_\phi(v_{\phi_i}) = -1$*;*
- *if* $\mathsf{unfold}(\phi_i) = \mathsf{X}\phi_j$*, then* $V_\phi = V_\phi \cup \{v_{\mathsf{X}\phi_j}\}$, $E_\phi = E_\phi \cup \{\langle v_{\mathsf{X}\phi_j}, v_{\phi_i}\rangle\}$, $\mathsf{W}_\phi(\langle v_{\mathsf{X}\phi_j}, v_{\phi_i}\rangle) = 1$, *and* $\mathsf{b}_\phi(v_{\phi_i}) = \mathsf{b}_\phi(v_{\mathsf{X}\phi_j}) = 0$*;*
- *if* $\mathsf{unfold}(\phi_i) = \phi_k \vee (\phi_j \wedge \mathsf{X}\phi_i)$*, then* $V_\phi = V_\phi \cup \{v_{\phi_k}, v_{\phi_j}, v_{\mathsf{X}\phi_i}\}$, $E_\phi = E_\phi \cup \{\langle v_{\phi_k}, v_{\phi_i}\rangle, \langle v_{\phi_j}, v_{\phi_i}\rangle, \langle v_{\mathsf{X}\phi_i}, v_{\phi_i}\rangle\}$, $\mathsf{W}_\phi(\langle v_{\phi_k}, v_{\phi_i}\rangle) = 2$, $\mathsf{W}_\phi(\langle v_{\phi_j}, v_{\phi_i}\rangle) = 1$, $\mathsf{W}_\phi(\langle v_{\mathsf{X}\phi_i}, v_{\phi_i}\rangle) = 1$, $\mathsf{b}_\phi(v_{\phi_i}) = -1$, *and* $\mathsf{b}_\phi(v_{\mathsf{X}\phi_i}) = 0$*,*

where $p \in \mathbb{P} \cup \{\top, \bot\}$ and ϕ_j, ϕ_k are LTL_f formulae.

Intuitively, the LTL_f graph of a formula construct the relation of satisfiability of the sub-formulae between current state and the next state. We use an example to illustrate the LTL_f graph.

Example 1. *Let $\phi = p\mathsf{U}(\mathsf{X}\neg q)$ be an LTL_f formula. We have $\mathsf{sub}(\phi) = \{p, q, \phi_3, \phi_4, \phi_5\}$, where $\phi_5 = p\mathsf{U}(\mathsf{X}\neg q)$, $\phi_4 = \mathsf{X}\neg q$, and $\phi_3 = \neg q$. The LTL_f graph of ϕ is illustrated as follows. $V_\phi = \{v_{\phi_5}, v_{\phi_4}, v_{\phi_3}, v_p, v_q, v_{\mathsf{X}\phi_5}, v_{\mathsf{X}\phi_3}\}$, $E_\phi = \{\langle v_{\mathsf{X}\phi_5}, v_{\phi_5}\rangle, \langle v_{\phi_4}, v_{\phi_5}\rangle, \langle v_p, v_{\phi_5}\rangle, \langle v_{\mathsf{X}\phi_3}, v_{\phi_4}\rangle, \langle v_q, v_{\phi_3}\rangle, \langle v_p, v_p\rangle, \langle v_q, v_q\rangle\}$, $\mathsf{W}_\phi(\langle v_{\mathsf{X}\phi_5}, v_{\phi_5}\rangle) = 1$, $\mathsf{W}_\phi(\langle v_{\phi_4}, v_{\phi_5}\rangle) = 2$, $\mathsf{W}_\phi(\langle v_p, v_{\phi_5}\rangle) = 1$, $\mathsf{W}_\phi(\langle v_{\mathsf{X}\phi_3}, v_{\phi_4}\rangle) = 1$, $\mathsf{W}_\phi(\langle v_q, v_{\phi_3}\rangle) = -1$, $\mathsf{W}_\phi(\langle v_p, v_p\rangle) = 1$, $\mathsf{W}_\phi(\langle v_q, v_q\rangle) = 1$, $\mathsf{b}_\phi(v_{\phi_5}) = -1$, $\mathsf{b}_\phi(v_{\phi_3}) = 1$, and $\mathsf{b}_\phi(v_{\phi_4}) = \mathsf{b}_\phi(v_p) = \mathsf{b}_\phi(v_q) = \mathsf{b}_\phi(v_{\mathsf{X}\phi_3}) = \mathsf{b}_\phi(v_{\mathsf{X}\phi_5}) = 0$.*

Next, we define the *state classifier* for any given trace.

Definition 4. *Let ϕ be an LTL_f formula such that $|\mathsf{sub}(\phi)| = L$, $(V_\phi, E_\phi, \mathsf{W}_\phi, \mathsf{b}_\phi)$ an LTL_f graph of ϕ, and $\pi = s_0, s_1, \ldots, s_n$ a trace. For each state $s_i \in \pi$, let*

$\mathbf{x}_{s_i} \in \mathbb{R}^L$ *represent a vector of the state* $s_i \in \pi$. $\mathbf{x}_{s_i}$ *is defined as a vector* $[x_1, \ldots, x_L]$ *one-to-one corresponding to* $\mathsf{sub}(\phi)$ *such that for all* $1 \leq i \leq L$, x_i *corresponds to* $\phi_i \in \mathsf{sub}(\phi)$ *and that for all* $1 \leq j < k \leq L, \phi_k \notin \mathsf{sub}(\phi_j)$. *Let* $\mathbf{x}_{s_i}^{(t)}$ *be the value of* $\mathbf{x}_{s_i}$ *at time* t. $\mathbf{x}_{s_i}^{(0)}$ *is defined such that for all* $1 \leq j \leq L, (\mathbf{x}_{s_i}^{(0)})_j = 1$ *if* $\phi_j \in s_i$ *or* $(\mathbf{x}_{s_i}^{(0)})_j = 0$ *otherwise.* $\mathbf{x}_{s_i}^{(t)}$ *is defined recursively as follows:*

$$\mathbf{x}_{s_i}^{(t)} = \sigma(\mathbf{C}_\phi \mathbf{x}_{s_i}^{(t-1)} + \mathbf{A}_\phi \mathbf{x}_{s_{i+1}} + \mathbf{b}_\phi), \tag{1}$$

where $t \geq 1$, $\sigma(x) = \min(\max(0, x), 1)$, *and* $\mathbf{C}_\phi, \mathbf{A}_\phi \in \mathbb{R}^{L \times L}$ *and* $\mathbf{b}_\phi \in \mathbb{R}^L$ *are parameters defined as follows:*

$$(\mathbf{C}_\phi)_{ij} = \begin{cases} \mathsf{W}_\phi(\langle v_{\phi_j}, v_{\phi_i} \rangle), & \text{if } \langle v_{\phi_j}, v_{\phi_i} \rangle \in E_\phi \text{ and } \phi_j \in \mathsf{sub}(\phi) \\ 0, & \text{otherwise,} \end{cases}$$

$$(\mathbf{A}_\phi)_{ij} = \begin{cases} \mathsf{W}_\phi(\langle v_{\phi_j}, v_{\phi_i} \rangle), & \text{if } \langle v_{\phi_j}, v_{\phi_i} \rangle \in E_\phi \text{ and } \phi_j \in \{\mathsf{X}\phi_k | \phi_k \in \mathsf{sub}(\phi)\} \\ 0, & \text{otherwise,} \end{cases}$$

$$(\mathbf{b}_\phi)_i = \mathsf{b}_\phi(v_{\phi_i}), \quad \text{for all } v_{\phi_i} \in V_\phi \text{ and } \phi_i \in \mathsf{sub}(\phi).$$

By $\mathcal{S}_\phi$ *we denote the* state classifier $\mathcal{S}_\phi$ *which accepts two vectors* $\mathbf{x}_{s_i}^{(0)}, \mathbf{x}_{s_{i+1}}$ *and a number of iterations* $T \in \mathbb{N}$ *as input, and outputs* $\mathbf{x}_{s_i}^{(T)}$, *i.e.,* $\mathbf{x}_{s_i}^{(T)} = \mathcal{S}_\phi(\mathbf{x}_{s_i}^{(0)}, \mathbf{x}_{s_{i+1}}, T)$.

Based on the state classifier, we define the *trace classifier*.

Definition 5. *Let* ϕ *be an LTL*$_f$ *formula such that* $|\mathsf{sub}(\phi)| = L$, *and* $\pi = s_0, s_1, \ldots, s_n$ *a trace. By* $\mathcal{T}_\phi$ *we denote the* trace classifier *which takes a vector* $\mathbf{x}_{s_i}^{(0)}$ *and a number of iterations* $T \in \mathbb{N}$ *as input and* $\mathbf{x}_{s_i}^{(T)}$ *as output; i.e.,* $\mathbf{x}_{s_i}^{(T)} = \mathcal{T}_\phi(\mathbf{x}_{s_i}^{(0)}, T)$, *where*

$$\mathcal{T}_\phi(\mathbf{x}_{s_i}^{(0)}, T) = \begin{cases} \mathcal{S}_\phi(\mathbf{x}_{s_i}^{(0)}, \mathcal{T}_\phi(\mathbf{x}_{s_{i+1}}^{(0)}, T), T), & 0 \leq i < n \\ \mathcal{S}_\phi(\mathbf{x}_{s_i}^{(0)}, \mathbf{0}, T), & i = n \end{cases} \tag{2}$$

Intuitively, for each state $s_i \in \pi$, $(\mathbf{x}_{s_i})_j$ represents the classifying result whether $\pi_i \models \phi_j$ where $\phi_j \in \mathsf{sub}(\phi)$. Specially, $(\mathbf{x}_{s_0})_L$ represents whether $\pi \models \phi$. Theorem 1 shows that the trace classifier can check the satisfaction of LTL$_f$ formulae over traces.

Theorem 1. *Let* ϕ *be an LTL*$_f$ *formula such that* $|\mathsf{sub}(\phi)| = L$. *For every trace* $\pi = s_0, s_1, \ldots, s_n$, $(\mathcal{T}_\phi(\mathbf{x}_{s_0}^{(0)}, L))_L = 1$ *if and only if* $\pi \models \phi$.

Intuitively, a state classifier is a simple homogeneous AC-GNN classifier, and a trace classifier distinguishes positive and negative traces by iteratively invoking the same simple homogeneous AC-GNN classifier. Learning LTL$_f$ formulae can be reduced to learning a simple homogeneous AC-GNN classifier that can distinguish positive and negative traces. Therefore, Theorem 1 provides a new idea to learn LTL$_f$ formulae: first learning a GNN classifier and then interpreting LTL$_f$ formulae from the GNN classifier. We will illustrate the approach in the next section. The formal proof of Theorem 1, as well as other formal statements, can be found in the technical report[1].

[1]Our code, benchmarks and technical report are publicly available at https://github.com/a79461378945/Bridging-LTLf-Inference-to-GNN-Inference-for-Learning-LTLf-Formulae.

GNN-based Learning Approach

In this section, we propose a GNN-based approach named `GLTLf` to learn LTL$_f$ formulae. In order to face the challenge about large search space, the key idea of `GLTLf` is to model the searching of a formula that potentially distinguish positive and negative traces by learning a simple homogeneous AC-GNN model. Thanks to the non-deterministic learning process, `GLTLf` can naturally handle noisy data. The framework of `GLTLf` is summarized as follows.

1. In the first phase, we train a simple homogeneous AC-GNN model $\mathcal{S}$ to distinguish positive and negative traces.
2. In the second phase, we interpret the parameters of $\mathcal{S}$ to obtain an LTL$_f$ formula ϕ_A.

Converting Traces to Graphs

To apply GNNs on traces, we convert traces to graphs. Let $\pi = s_0, s_1, \ldots, s_n$ be a trace and $\mathbb{P}$ a set of atomic propositions. We convert π to a directed path graph $G_\pi = (V_\pi, E_\pi)$, where V_π is the set of vertices and E_π is the set of edges. Each vertex v_i in V_π corresponds to a state s_i in π. For each pair of adjacent states s_i, s_{i+1} in π, there is a corresponding edge $\langle v_{i+1}, v_i \rangle$ in E_π.

Training GNNs as Classifiers

Let ϕ be a formula to be learned. To apply GNN on graph G_π, each vertex $v_i \in V_\pi$ has an associated feature vector $\mathbf{x}_{v_i} \in \mathbb{R}^L$ represented as $[x_1, \ldots, x_{|\mathbb{P}|}, x_\top, x_{|\mathbb{P}|+2}, \ldots, x_L]$. We set $L = |\mathbb{P}| + 1 + k_g$, where k_g is a hyperparameter representing the number of sub-formulae of non-atomic propositions. For all $1 \leq i \leq L$, x_i corresponds to ϕ_i and for all $1 \leq j < k \leq L$, $\phi_k \notin \mathsf{sub}(\phi_j)$ where ϕ_i, ϕ_j and ϕ_k are possible sub-formulae of ϕ. Specially, $x_1, \ldots, x_{|\mathbb{P}|}$ one-to-one correspond to propositions in $\mathbb{P}$, $x_\top$ corresponds to $\top$, and x_L corresponds to ϕ. $(\mathbf{x}_{v_i})_j$ means the classifying result whether $\pi_i \models \phi_j$ where $\phi_j \in \mathsf{sub}(\phi)$. Let $\mathbf{x}_{v_i}^{(t)}$ be the value of $\mathbf{x}_{v_i}$ at the t^{th} iteration. If $\phi_j \in s_i$, then ϕ_j is a proposition and $s_i \models \phi_j$. Therefore, $\mathbf{x}_{v_i}^{(0)}$ is defined such that for all $1 \leq j \leq L, (\mathbf{x}_{v_i}^{(0)})_j = 1$ if $\phi_j \in s_i$ or $(\mathbf{x}_{v_i}^{(0)})_j = 0$ otherwise.

The state classifier $\mathcal{S}$ follows Definition 4, but the parameters $\mathbf{C}, \mathbf{A} \in \mathbb{R}^{L \times L}$ and $\mathbf{b} \in \mathbb{R}^L$ need to be trained. Formally, $\mathcal{S}$ accepts two vectors $\mathbf{x}_{v_i}^{(0)}, \mathbf{x}_{v_{i+1}}$ and a number of iterations $T \in \mathbb{N}$ as input, and outputs $\mathbf{x}_{v_i}^{(T)}$. $\mathcal{S}$ iteratively updates the vector $\mathbf{x}_{v_i}$ by passing the satisfaction relation between π_{i+1} and each sub-formulae of ϕ forth along the edges of G_π:

$$\begin{aligned} \mathbf{x}_{v_i}^{(t)} &= \mathsf{COM}(\mathbf{x}_{v_i}^{(t-1)}, \mathsf{AGG}(\{\{\mathbf{x}_u^{(t-1)} | u \in \mathcal{N}(v_i)\}\})) \\ &= \sigma(\mathbf{C}\mathbf{x}_{v_i}^{(t-1)} + \mathbf{A}\mathbf{x}_{v_{i+1}} + \mathbf{b}), \end{aligned} \tag{3}$$

There is at most one neighbor in each vertex in the path graph and the vector of the neighbor is not updated, thus $\mathsf{AGG}(\{\{\mathbf{x}_u^{(t-1)} | u \in \mathcal{N}(v_i)\}\}) = \mathbf{x}_{v_{i+1}}$ if $0 \leq i < n$ or $\mathbf{0}$ otherwise. To overcome the problem of vanishing gradient, we set σ to a variant of leaky ReLU defined by:

$$\sigma(x) = \begin{cases} \alpha x, & x < 0, \\ x, & 0 \leq x \leq 1, \\ \alpha x + 1 - \alpha, & x > 1, \end{cases} \tag{4}$$

where α is a hyperparameter. The trace classifier $\mathcal{T}$ is defined by Definition 5. $(\mathcal{T}(\mathbf{x}_{v_i}^{(0)}, L))_j$ indicates whether $\pi_i \models \phi_j$, where $\phi_j \in \mathsf{sub}(\phi)$. Specially, $(\mathcal{T}(\mathbf{x}_{v_0}^{(0)}, L))_L$ indicates whether $\pi \models \phi$.

The classification objective are formulated as:

$$\zeta_1 = ((\mathcal{T}(\mathbf{x}_{v_0}^{(0)}, L))_L - \phi_T(\pi))^2. \tag{5}$$

To obtain sparse parameters that are suitable for interpreting formulae from, we apply L1 regularization to $\mathbf{C}$ and $\mathbf{A}$. The regularization is formulated as:

$$\zeta_2 = \sum_{i=1}^{L}\sum_{j=1}^{L}|(\mathbf{C})_{ij}| + \sum_{i=1}^{L}\sum_{j=1}^{L}|(\mathbf{A})_{ij}|. \tag{6}$$

We force the range of the parameters of $\mathbf{C}$ to be $[-1, 2]$ and the range of the parameters of $\mathbf{A}$ to be $[0, 1]$. The range of the parameters comes from Definition 3 and is for learning interpretable parameters. The regularization is formulated as:

$$\begin{aligned}\zeta_3 &= \sum_{i=1}^{L}\sum_{j=1}^{L}(\mathsf{Relu}((\mathbf{C})_{ij} - 2) + \mathsf{Relu}(-(\mathbf{C})_{ij} - 1)),\\ \zeta_4 &= \sum_{i=1}^{L}\sum_{j=1}^{L}(\mathsf{Relu}((\mathbf{A})_{ij} - 1) + \mathsf{Relu}(-(\mathbf{A})_{ij})).\end{aligned} \tag{7}$$

The final joint objective is:

$$\zeta = \zeta_1 + \beta\zeta_2 + \gamma(\zeta_3 + \zeta_4), \tag{8}$$

where β, γ are the regularization coefficients. We train $\mathcal{S}$ to minimize the joint objective.

In order to learn interpretable parameters, We freeze parts of the parameters in $\mathbf{C}, \mathbf{A}, \mathbf{b}$ when training. Specifically, we freeze $\mathbf{C}_i, \mathbf{A}_i, \mathbf{b}_i$ for i from 1 to $|\mathbb{P}| + 1$ because they correspond to atomic propositions. For i from $|\mathbb{P}| + 2$ to L, we freeze $\mathbf{C}_{ij}$ for j from i to L and $\mathbf{A}_{ij}$ for j from $i + 1$ to L. It is to avoid the situation where the two formulae are sub-formulas of each other.

Interpreting LTL$_f$ Formulae from GNNs

We interpret an LTL$_f$ formulae ϕ_A from the parameters $\mathbf{C}, \mathbf{A}, \mathbf{b}$ of a learned GNNs. Based on the equation 3, $(\mathbf{x}_{v_j}^{(t)})_i$ depends on $(\mathbf{x}_{v_j}^{(t-1)})_i$ and $(\mathbf{x}_{v_{j+1}})_i$. $(\mathbf{C})_i, (\mathbf{A})_i, (\mathbf{b})_i$ describe the their dependence which is like the $\mathsf{W}_{\phi_T}, \mathsf{b}_{\phi_T}$ of LTL$_f$ graph of the target formula ϕ_T (Definition 3). Therefore, for any $1 \le i \le L$, we interpret a sub-formula from $(\mathbf{C})_i$, $(\mathbf{A})_i$, and $(\mathbf{b})_i$. We call the interpreted sub-formula as *interpretation*. Since for any $1 \le i \le |\mathbb{P}|$ $(\mathbf{x})_i$ corresponds to an atomic proposition and $(\mathbf{x})_{|\mathbb{P}|+1}$ corresponds to $\top$, we only interpret the sub-formulae for any $|\mathbb{P}| + 2 \le i \le L$.

Because the model training by gradient descent results to uncontrollable bias of parameters for interpreting, we only interpret an approximate formula. The main idea is to first recommend a set of candidate formulae based on a cheap metric that measures the rationality of interpretations, and then select the best formula based on some expensive metrics, such as the discrimination effect for the traces. To this end, we define *interpretation similarity* between an interpretation and a sub-formula interpreted by parameters.

Definition 6. *Let ϕ_i be an LTL$_f$ formula and $\mathbf{C}, \mathbf{A}, \mathbf{b}$ parameters. The* interpretation similarity *between ϕ_i and the interpretation of $(\mathbf{C})_i, (\mathbf{A})_i, (\mathbf{b})_i$ is defined as follows:*

$$\mathsf{sim}(\phi_i, (\mathbf{A})_i, (\mathbf{C})_i, (\mathbf{b})_i) = \begin{cases} \frac{1}{1+\mathsf{dis}([(\mathbf{C})_{ij}, (\mathbf{b})_i], [-1,1])}, & \phi_i = \neg\phi_j,\\ \frac{1}{1+\mathsf{dis}([(\mathbf{A})_{ij}], [1])}, & \phi_i = \mathsf{X}\phi_j,\\ \frac{1}{1+\mathsf{dis}([(\mathbf{C})_{ij}, (\mathbf{C})_{ik}, (\mathbf{b})_i], [1,1,-1])}, & \phi_i = \phi_j \wedge \phi_k,\\ \frac{1}{1+\mathsf{dis}([(\mathbf{C})_{ij}, (\mathbf{C})_{ik}, (\mathbf{A})_{ii}, (\mathbf{b})_i], [1,2,1,-1])}, & \phi_i = \phi_j \mathsf{U} \phi_k \end{cases}$$

where $\mathsf{dis}(v_1, v_2)$ is the euclidean distance between the vector v_1 and the vector v_2.

Intuitively, the interpretation similarity measures the difference between an interpretation and the sub-formula interpreted by parameters. For example, if the interpretation of ϕ_i is $\phi_j \wedge \phi_k$, then $(\mathbf{C})_{ij}$ is close to 1; $(\mathbf{C})_{ik}$ is close to 1; $(\mathbf{b})_i$ is close to -1. These values that need to be close are from Definition 3. The greater the degree of approximation, the better the interpretation. The total interpretation similarity of a formula ϕ is the product of the interpretations of all sub-formulae of ϕ.

Algorithm 1: Interpreting LTL$_f$ Formulae

Input : A training set Π, parameters of GNNs $\mathbf{C}, \mathbf{A}, \mathbf{b}$, and the number of best interpretations θ_{ik}.
Output : A formula ϕ.

1 $\forall i = 1, 2, \ldots, |\mathbb{P}| + 1, f_i \leftarrow (1, \emptyset)$;
2 $\forall i = |\mathbb{P}| + 2, |\mathbb{P}| + 3, \ldots, L, f_i \leftarrow \emptyset$;
3 **for** $i \leftarrow |\mathbb{P}| + 2$ *to* L **do**
4 **for** $\phi_i \in \{\neg\phi_j | j < i \wedge (\mathbf{C})_{ij} < 0\} \cup \{\mathsf{X}\phi_j | j < i \wedge (\mathbf{A})_{ij} > 0\}$ **do**
5 **for** $(s_j, c_j) \in f_j$ **do**
6 $f_i \leftarrow f_i \cup \{(\mathsf{sim}(\phi_i, (\mathbf{A})_i, (\mathbf{C})_i, (\mathbf{b})_i) \cdot s_j, \{(i, \phi_i)\} \cup c_j)\}$;
7 **for** $\phi_i \in \{\phi_j \wedge \phi_k | j, k < i \wedge (\mathbf{C})_{ij}, (\mathbf{C})_{ik} > 0\} \cup \{\phi_j \mathsf{U} \phi_k | j, k < i \wedge (\mathbf{A})_{ii}, (\mathbf{C})_{ij}, (\mathbf{C})_{ik} > 0\}$ **do**
8 **for** $(s_j, c_j) \in f_j, (s_k, c_k) \in f_k$ **do**
9 $f_i \leftarrow f_i \cup \{(\mathsf{sim}(\phi_i, (\mathbf{A})_i, (\mathbf{C})_i, (\mathbf{b})_i) \cdot s_j \cdot s_k, \{(i, \phi_i)\} \cup c_j \cup c_k)\}$;
10 sort f_i according to the total interpretation similarity and preserve the top θ_{ik} elements in f_i;
11 construct top θ_{ik} best candidate formulae Φ from ϕ_L;
12 $\phi_A \leftarrow$ select the best formula from Φ according to some priorities;
13 **return** ϕ_A;

Algorithm 1 shows the process of interpreting. We interpret every sub-formula from bottom to top, *i.e.* interpret ϕ_i, where i is from $|\mathbb{P}| + 2$ to L (lines 3-10). f_i denotes the set of total interpretations of ϕ_i, which includes the interpretations of ϕ_i and its sub-formulae. $(s_i, c_i) \in f_i$ denotes the tuple of total interpretation similarity of ϕ_i (s_i) and a set of

interpretations of sub-formulae of ϕ_i (c_i). $(j, \phi_j) \in c_i$ denotes the interpretation of ϕ_j. Intuitively, we can construct the entire formula ϕ_i from top to bottom by traversing c_i. For each ϕ_i, we first enumerate all possible interpretations of ϕ_i and update f_i (lines 4-9). Then, we sort f_i according to the total interpretation similarity to preserve the top θ_{ik} best total interpretations for ϕ_i (line 10). Finally, we obtain the top θ_{ik} best candidate formulae Φ from ϕ_L (line 11). We ranks the candidate formulae according to the priorities: (1) the accuracy of traces classification; (2) the size of formulae; (3) the interpretation distance. We select the best formula ϕ_A as the result of interpreting (line 12). We use an example to illustrate Algorithm 1.

i	ϕ_i	sim	total sim	f_i
1	ϕ_1	1	**1**	$\{(1, \emptyset)\}$
2	ϕ_2	1	**1**	$\{(1, \emptyset)\}$
3	$\neg\phi_2$	1	**1**	$\{(1, \{(3, \neg\phi_2)\})\}$
4	$\mathsf{X}\phi_2$	0.56	**0.56**	$\{(1, \{(4, \mathsf{X}\phi_3), (3, \neg\phi_2)\}),$ $(0.56, \{(4, \mathsf{X}\phi_2)\})\}$
	$\mathsf{X}\phi_3$	1	**1**	
5	$\phi_1 \wedge \phi_3$	0.77	**0.77**	$\{(0.83, \{(5, \phi_1 \mathsf{U} \phi_4),$ $(4, \mathsf{X}\phi_3), (3, \neg\phi_2)\}),$ $(0.77, \{(5, \phi_1 \wedge \phi_3),$ $(3, \neg\phi_2)\})\}$
	$\phi_1 \wedge \phi_4$	0.56	0.31 0.56	
	$\phi_3 \wedge \phi_4$	0.54	0.30 0.54	
	$\phi_1 \mathsf{U} \phi_3$	0.43	0.43	
	$\phi_1 \mathsf{U} \phi_4$	0.83	0.46 **0.83**	
	$\phi_3 \mathsf{U} \phi_1$	0.49	0.49	
	$\phi_3 \mathsf{U} \phi_4$	0.73	0.41 0.73	
	$\phi_4 \mathsf{U} \phi_1$	0.44	0.41 0.44	
	$\phi_4 \mathsf{U} \phi_3$	0.40	0.22 0.40	

Table 1: The process of Algorithm 1 on Example 2. The order in each row related to ϕ_4 of column total sim is the same as the row where $i = 4$. The numbers in **bold** denote the θ_{ik} elements in f_i. f_i is shown after preservation.

Example 2. *Assume* $|\mathbb{P}| = 2$, $\theta_{ik} = 2$, *and* $\mathbf{C}, \mathbf{A}, \mathbf{b}$ *are as follows, where we remove* $(\mathbf{x})_{|\mathbb{P}|+1}$ *corresponding to* $\top$ *for simplification.*

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0.4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0.7 & 1.8 & 0 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ -1 \end{bmatrix}$$

First, $f_1, f_2 \leftarrow \{(1, \emptyset)\}$, $f_3, f_4, f_5 \leftarrow \emptyset$. *Then, we calculate* f_i *for each* i *from 3 to 5. The enumerations in each iteration are shown in Table 1. Finally, we construct top* θ_{ik} *best candidate formulae* Φ *from* ϕ_L. *Take the first element in* ϕ_L, $(0.83, \{(5, \phi_1 \mathsf{U} \phi_4), (4, \mathsf{X}\phi_3), (3, \neg\phi_2)\})$, *for an example. It represents that* $\phi_5 = \phi_1 \mathsf{U} \phi_4$, $\phi_4 = \mathsf{X}\phi_3$, $\phi_3 = \neg\phi_2$. *We only use the atomic propositions* ϕ_1, ϕ_2 *to represent* ϕ_5, *so* $\phi_5 = \phi_1 \mathsf{U} \mathsf{X} \neg \phi_2$. *We select the best formula* ϕ_A *according to the priorities.*

Preliminary Results

To evaluate our approach, we considered three research questions as follows. (1) Is `GLTLf` better than state-of-the-art (SOTA) approaches in noisy data or noise-free data? (2) What is the performance of GNN model and interpreting method? (3) How about the scalability of the size of formulae and the robustness of hyperparameter k_g?

Competitors. We compared `GLTLf` with three SOTA approaches in different setting shown in Table 2.

approach	noise	arbitrary
`C.&M.` (Camacho and McIlraith 2019)	×	✓
`BayesLTL` (Kim et al. 2019)	✓	×
`MaxSAT-DT` (Gaglione et al. 2021)	✓	✓
`GLTLf` (Ours)	✓	✓

Table 2: Details about SOTA approaches. "noise" means noisy data. "arbitrary" means arbitrary formulae.

Datasets. We used randltl tool of SPOT (Duret-Lutz et al. 2016) to generate random formulae. The tool generated unique formulae following a specified token distribution ($\mathbb{P}$, $\mathbb{B}$, standard logical operators, temporal modal operators) in a specified size interval. The number of different atomic propositions was fixed to 5. Our token distribution put weight as follows, $\mathbb{P}$: 2.5, $\top$: 1, $\neg$: 1, $\vee$: 1, X: 1, and U: 1. The propositions in $\mathbb{P}$ obeyed uniform distribution. We generated 5 domains for $k_f \in \{3, 6, 9, 12, 15\}$. For each domain, we generated 50 datasets. For each dataset, we generated a formula of which has k_f sub-formulae of non-atomic propositions and then we randomly generated $250/250$ positive/negative traces of this formula as the training set, $500/500$ positive/negative traces of this formula as the test set.

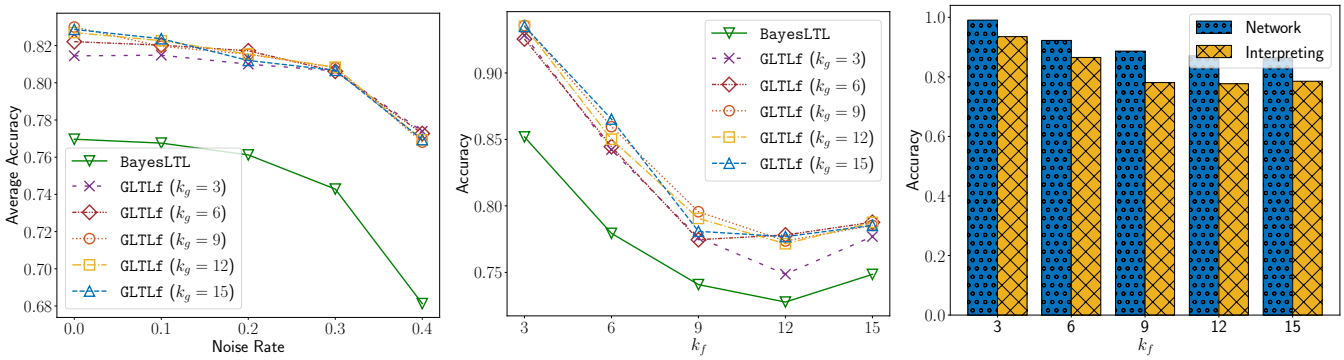
To generate a trace, we first determined the length of the trace by an random integer between 1 and 20. Then for each state in the trace, each atom proposition was set true with a probability of $\frac{1}{3}$. For each trace in the set, we used a symbolic path checking algorithm to decide whether it is a positive trace or a negative one. If there ware not enough positive/negative traces, we generated more random traces and continue the procedure.

To generate noisy data, following the work (Kim et al. 2019; Gaglione et al. 2021), we randomly chose some traces from the original data and give them wrong labels (if it was a positive label, it becomes a negative label, and vice versa). The number of traces with wrong labels is determined by the noise rate δ. For example, if the noise rate is 0.1, 10% of the traces will be given wrong labels.

Settings. All experiments are run on a Linux equipped with an Intel(R) Xeon(R) Gold 5218R processor with 2.1 GHz and 126 GB RAM. The time limit is 1 hour and the memory limit is 10 GByte for each instance. Hyperparameters of `GLTLf` are set as follows: $\theta_{ik} = 100$, $k_g \in \{3, 6, 9, 12, 15\}$

	$k_f = 3$			$k_f = 6$			$k_f = 9$			$k_f = 12$			$k_f = 15$		
	Acc	F1	N_s	Acc	F1	N_s	Acc	F1	N_s	Acc	F1	N_s	Acc	F1	N_s
C.&M.	99.77	99.77	50	97.93	96.79	47	97.14	95.55	35	95.10	91.91	20	93.74	87.49	8
BayesLTL	85.19	85.96	50	77.94	76.78	50	74.08	75.73	50	72.77	73.47	50	74.85	77.32	50
MaxSAT-DT	100	100	49	100	100	19	100	100	8	100	100	5	100	100	5
GLTLf	93.50	93.09	50	86.53	86.28	50	78.09	78.66	50	77.69	78.63	50	78.53	79.34	50

Table 3: Experiment results at $k_g = 15$ on noise-free data across different approaches. "Acc" means accuracy (%). "F1" means F1 score (%). "N_s" stands for the number of successfully solved formulae (50 total).



(a) (b) (c)

Figure 1: (a) Accuracies among different noise rates. The results are the average results of the 5 accuracy when $k_f \in \{3, 6, 9, 12, 15\}$. (b) Accuracies among different k_g settings. There results are from noise-free datasets. (c) Net accuracy and interpreting accuracy. There results are from $k_g = 15$ setting and noise-free datasets.

to study the robustness of k_g, $\alpha = 0.1$, $\beta = 0.1$, and $\gamma = 10$. We use Adam (Kingma and Ba 2015) to optimize the parameters in our model. The learning rate is set to $1e-4$. The number of epochs is 400 and the batch size is 10. The number of samples for BayesLTL is set to 2M and other settings for BayesLTL are default. In experiments, all approaches first learn an LTL$_f$ formula from the training set and then are compared by evaluating the classification effect of the learned formulae on the test set.

Result Analysis

The results shown below are the average results on 50 different datasets in each domain.

Comparisons on noise-free data. As shown in Table 3, our approach significantly surpasses BayesLTL. Although MaxSAT-DT and C.&M. are in the lead, they cannot solve long formulae. MaxSAT-DT and C.&M. fail to solve most formulae (due to time-out) as k_f increased.

Comparisons on noisy data. Our approach is proven to be more noise-tolerated then other approaches by Figure 1(a). We evaluate all compared approaches on noisy datasets with $\delta \in \{0.1, 0.2, 0.3, 0.4\}$. Figure 1(a) shows that GLTLf surpasses BayesLTL notably. MaxSAT-DT and C.&M. also fail to solve all formulae when the training data was noisy.

Performance of interpreting. Our approach consists of network training and interpreting. Thus, we studied the performance gap between both parts. As shown in Figure 1(c), there is a significant gap between net accuracy and interpreting accuracy. The gap suggests that there is a lot of potential for the interpreting method to evolve.

Scalability and robustness. Table 3 and Figure 1(b) show that our approach can handle long formulae because our approach gets all the results with higher performance compared to other approaches. We also studied the robustness of network size. As shown in Figure 1(b), larger networks achieve better performance, even when solving short formulae, while small networks can also achieve comparable performance on long formulae to larger networks, such as GLTLf ($k_g = 6$). Meanwhile, we find that the accuracy decreases with increasing formulae complexity k_g except for $k_g = 15$. The reason might be that the formula with $k_g = 15$ is more likely to be simplified. In summary, our approach is able to solve formulae in various sizes and is rather robust.

Conclusions and Future Work

Learning LTL$_f$ formulae to characterize the high-level behavior of a system is an important and challenging problem in planning. We theoretically bridge LTL$_f$ inference to GNN inference, which provides a new method for learning arbitrary LTL$_f$ formulae from noisy data. Based on the theoretical result, we design a GNN-based approach, named as GLTLf. Experimental results demonstrate that our approach is stronger robustness for noisy data and better scalability in data size.

Future work will extend our approach to learning LTL$_f$ with uncertainty and explore interpretable GNN learning.

Acknowledgments

We thank Rongzhen Ye for discussion on the paper and anonymous referees for helpful comments. This paper was supported by the National Natural Science Foundation of China (No. 61976232 and 61876204), Guangdong Basic and Applied Basic Research Foundation (No.2022A1515011355 and 2020A1515010642), Guizhou Science Support Project (No. 2022-259), Humanities and Social Science Research Project of Ministry of Education (18YJCZH006).

References

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2): 87–106.

Angluin, D.; Eisenstat, S.; and Fisman, D. 2015. Learning Regular Languages via Alternating Automata. In *IJCAI*, 3308–3314.

Arif, M. F.; Larraz, D.; Echeverria, M.; Reynolds, A.; Chowdhury, O.; and Tinelli, C. 2020. SYSLITE: syntax-guided synthesis of PLTL formulas from finite traces. In *FMCAD*, 93–103.

Asarin, E.; Donzé, A.; Maler, O.; and Nickovic, D. 2011. Parametric identification of temporal properties. In *RV*, 147–160.

Baier, J. A.; and McIlraith, S. A. 2006. Planning with Temporally Extended Goals Using Heuristic Search. In *ICAPS*, 342–345.

Barceló, P.; Kostylev, E. V.; Monet, M.; Pérez, J.; Reutter, J. L.; and Silva, J. P. 2020. The Logical Expressiveness of Graph Neural Networks. In *ICLR*, 1–21.

Bollig, B.; Habermehl, P.; Kern, C.; and Leucker, M. 2009. Angluin-Style Learning of NFA. In *IJCAI*, 1004–1009.

Bombara, G.; Vasile, C. I.; Penedo, F.; Yasuoka, H.; and Belta, C. 2016. A Decision Tree Approach to Data Classification using Signal Temporal Logic. In *HSCC*, 1–10.

Brunello, A.; Sciavicco, G.; and Stan, I. E. 2020. Interval Temporal Logic Decision Tree Learning. *CoRR*, abs/2003.04952.

Camacho, A.; and McIlraith, S. A. 2019. Learning Interpretable Models Expressed in Linear Temporal Logic. In *ICAPS*, 621–630.

Cao, Z.; Tian, Y.; Le, T. B.; and Lo, D. 2018. Rule-based specification mining leveraging learning to rank. *Automated Software Engineering*, 25(3): 501–530.

Ciccio, C. D.; Mecella, M.; and Mendling, J. 2013. The Effect of Noise on Mined Declarative Constraints. In *SIMPDA*, 1–24.

Duret-Lutz, A.; Lewkowicz, A.; Fauchille, A.; Michaud, T.; Renault, E.; and Xu, L. 2016. Spot 2.0 - A Framework for LTL and ω -Automata Manipulation. In *ATVA*, 122–129.

Gaglione, J.; Neider, D.; Roy, R.; Topcu, U.; and Xu, Z. 2021. Learning Linear Temporal Properties from Noisy Data: A MaxSAT Approach. *CoRR*, abs/2104.15083.

Giacomo, G. D.; and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, 854–860.

Giantamidis, G.; and Tripakis, S. 2016. Learning Moore Machines from Input-Output Traces. In *FM*, 291–309.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Engineering Bulletin*, 40(3): 52–74.

Jin, X.; Donzé, A.; Deshmukh, J. V.; and Seshia, S. A. 2015. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11): 1704–1717.

Kasenberg, D.; and Scheutz, M. 2017. Interpretable apprenticeship learning with temporal logic specifications. In *CDC*, 4914–4921.

Kim, J.; Muise, C.; Shah, A.; Agarwal, S.; and Shah, J. 2019. Bayesian Inference of Linear Temporal Logic Specifications for Contrastive Explanations. In *IJCAI*, 5591–5598.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*, 1–15.

Kong, Z.; Jones, A.; and Belta, C. 2016. Temporal logics for learning and detection of anomalous behavior. *IEEE Transactions on Automatic Control*, 62(3): 1210–1222.

la Rosa, T. D.; and McIlraith, S. A. 2011. Learning Domain Control Knowledge for TLPlan and Beyond. In *ICAPS-11 workshop PAL*, 1–8.

Le, T. B.; and Lo, D. 2015. Beyond support and confidence: Exploring interestingness measures for rule-based specification mining. In *SANER*, 331–340.

Lemieux, C.; Park, D.; and Beschastnikh, I. 2015. General LTL Specification Mining (T). In *ASE*, 81–92.

Leno, V.; Dumas, M.; Maggi, F. M.; Rosa, M. L.; and Polyvyanyy, A. 2020. Automated discovery of declarative process models with correlated data conditions. *Information Systems*, 89: 101482.

Maggi, F. M.; Bose, R. P. J. C.; and van der Aalst, W. M. P. 2012. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In *CAiSE*, 270–285.

Maggi, F. M.; Ciccio, C. D.; Francescomarino, C. D.; and Kala, T. 2018. Parallel algorithms for the automated discovery of declarative process models. *Information Systems*, 74: 136–152.

Maggi, F. M.; Montali, M.; and Peñaloza, R. 2020. Temporal Logics Over Finite Traces with Uncertainty. In *AAAI*, 10218–10225.

Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; and Grohe, M. 2019. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *AAAI*, 4602–4609.

Neider, D.; and Gavran, I. 2018. Learning Linear Temporal Properties. In *FMCAD*, 1–10.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*, 46–57.

Roy, R.; Fisman, D.; and Neider, D. 2020. Learning Interpretable Models in the Property Specification Language. In *IJCAI*, 2213–2219.

Shah, A.; Kamath, P.; Shah, J. A.; and Li, S. 2018. Bayesian Inference of Temporal Task Specifications from Demonstrations. In *NeurIPS*, 3808–3817.

Smetsers, R.; Fiterau-Brostean, P.; and Vaandrager, F. W. 2018. Model Learning as a Satisfiability Modulo Theories Problem. In *LATA*, 182–194.

Xu, K.; Hu, W.; Leskovec, J.; and Jegelka, S. 2019a. How Powerful are Graph Neural Networks? In *ICLR*, 1–17.

Xu, Z.; Birtwistle, M.; Belta, C.; and Julius, A. 2016. A temporal logic inference approach for model discrimination. *IEEE life sciences letters*, 2(3): 19–22.

Xu, Z.; and Julius, A. A. 2016. Census signal temporal logic inference for multiagent group behavior analysis. *IEEE Transactions on Automation Science and Engineering*, 15(1): 264–277.

Xu, Z.; Nettekoven, A. J.; Julius, A. A.; and Topcu, U. 2019b. Graph Temporal Logic Inference for Classification and Identification. In *CDC*, 4761–4768.